Module 3: Ordinateur et processeur (suite)

1 Introduction

Nous avons vu dans la leçon précédente l'architecture de Von Neumann et ses différents composants à savoir l'unité arithmétique et logique (UAL), l'unité de contrôle, la mémoire, et les dispositifs d'entrées/sorties. Nous avons également parlé des instructions, leurs types et formats.

Dans cette leçon, nous allons présenter les modes d'adressages et les différents registres du processeur. Nous finirons cette leçon par présenter la différence entre les d'architectures RISC vs CISC.

2 Modes d'adressage

Plusieurs modes d'adressage existe pour les instructions au sein d'un même processeur. La variété des modes d'adressage simplifie énormément les accès aux différentes structures de données utilisées par les langages de programmation évolués. Ce qui facilitera au compilateur la tâche de traduction des programmes en code assembleur.

2.1 Adressage immédiat

Dans ce mode d'adressage, les données sont spécifiées directement dans l'instruction. Par exemple, nous pouvons écrire en langage assembleur le code suivant: ADD r1, r2, #8 qui signifie additionner la valeur 8 au registre r2 et enregistre le résultat dans le registre r1. Dans ce cas, nous avons spécifié explicitement la données dans l'instruction au lieu d'aller la chercher dans le registre ou dans la mémoire.

Cependant, l'inconvénient de l'adressage immédiat réside dans le fait de manipuler des constantes uniquement. C'est à dire, #8 est une constante que nous voulons additionner avec le registre r2 dans l'exemple précédent. Par conséquent, ces données ne doivent pas dépendre d'un calcul antérieur.

2.2 Adressage direct

Si nous connaissons l'adresse mémoire où la donnée est stockée, nous pouvons spécifier l'adresse mémoire directement dans l'instruction. Par exemple, si la donnée est stockée dans l'adresse mémoire 9aff (hexadécimale), nous pouvons

donc écrire: ADD r1, r2, 9aff. Notez que l'adresse mémoire ici n'est pas précédée par le dièse (#). Cela signifie que le processeur va chercher la donnée dans l'adresse mémoire $9aff_{16}$.

Il est clair que ce type d'adressage présente un inconvénient qui est la connaissance préalable des adresses mémoires où les données sont stockées.

2.3 Adressage par registre

C'est le type généralement utilisé dans les programmes d'assemblage. Car les registres du processeur représentent l'un des endroits où les données peuvent être stockées. Donc, les registres sont donc désignés comme source ou destination dans les instruction comme nous l'avons vu dans les exemples du code assembleur vus précédemment.

2.4 Adressage indirect par registre

Nous avons vu dans le cas d'adressage direct, que nous pouvons indiquer l'adresse mémoire dans l'instruction. Nous pouvons également indiquer où se trouve cette adresse, dans un registre et donc le processeur va chercher l'adresse dans ce registre. Par exemple, nous pouvons écrire en assembleur le code suivant: ADD r1, (r2) qui signifie additionner la donnée se trouvant dans le registre r1 avec la donnée se trouvant dans la mémoire à l'adresse spécifié dans le registre r2. L'adressage indirect par registre est illustré dans la figure suivante[1].

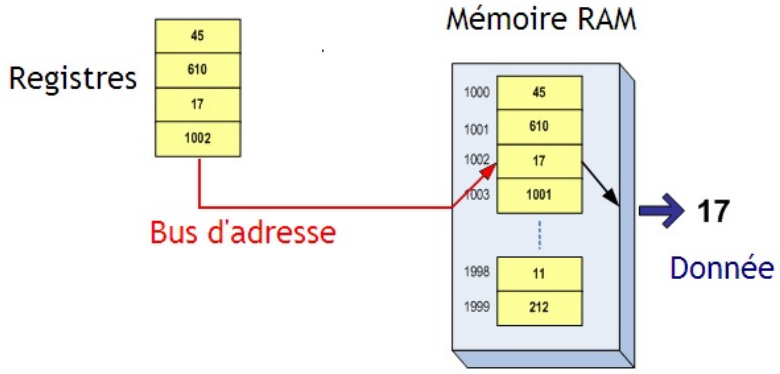

FIGURE 1 – Exemple d'adressage indirect par registre.

2.5 Adressage indirect avec déplacement

Ce mode d'adressage est une combinaison du mode d'adressage indirect avec un décalage. L'adresse de la donnée est obtenue en additionnant l'adresse contenue dans un registre (adressage indirect) et la valeur du déplacement immédiat

[1] Figure tirée du Wikipédia `https://fr.wikipedia.org/wiki/Mode_d%27adressage`.

indiquée dans l'instruction. Par exemple, nous pouvons écrire en assembleur l'instruction suivante: ADD r1, 5(r2) qui signifie additionner la donnée se trouvant dans le registre 1 et la donnée se trouvant dans la mémoire à l'adresse obtenue par la somme de 5 et le contenu du registre r2.

2.6 Adressage indirect avec post-incrémentation ou pré-décrémentation

L'adressage indirect est largement utilisé dans l'adressage des tableaux. À cet effet, deux autres types d'adressage, basés sur l'adressage indirect, sont introduits:

- **Avec post-incrémentation**: dans ce type d'adressage, l'instruction peut automatiquement augmenter le contenu du registre par une valeur fixe. Cela facilitera le parcours des tableaux de début jusqu'à la fin. La figure 2 présente un exemple d'adressage indirect avec post-incrémentation[2].

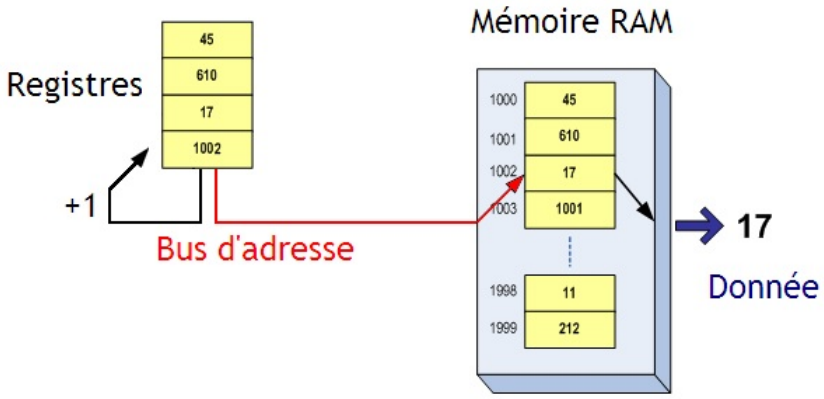

FIGURE 2 – Exemple d'adressage indirect avec post-incrémentation.

- **Avec pré-incrémentation**: dans ce type d'adressage, l'instruction peut automatiquement diminuer le contenu du registre par une valeur fixe. De même, ce type d'adressage facilitera le parcours des tableaux dans le sens inverse.

2.7 Adressage indirect indexé

Nous avons vu dans l'adressage indirect avec déplacement que la valeur de déplacement doit être spécifiée (connu d'avance), ce qui constitue un inconvénient de ce type d'adressage. Pour faire face à ce problème, l'adressage indexé est introduit. Dans cet adressage, la valeur de déplacement est stockée dans un

[2]Figure tirée du Wikipédia `https://fr.wikipedia.org/wiki/Mode_d%27adressage`.

registre au lieu d'être codée manuellement dans l'instruction. Par conséquent, l'adresse mémoire de la donnée est la somme du contenu d'un registre de base (mode indirect) et d'un registre jouant le rôle d'un déplacement. Nous pouvons ainsi écrire en assembleur: ADD r1, (r2, r3)

3 Registres

Nous avons mentionné dans les sections précédentes les registres et comment ils sont utilisés pour stocker des données et des adresses mémoires. Dans cette section, nous allons présenter les différents types de registres et leurs rôle dans l'exécution des instruction.

Les registres représentent des unités de stockage temporaire dans le processeur. Ils sont construits à l'aide de circuits logiques séquentiels pour pouvoir mémoriser des bits. L'intérêt des registres est de pouvoir travailler avec des données se trouvant directement dans le processeur, ce qui assure un accès beaucoup plus rapide que les données stockées en mémoire principale [3].

3.1 Les registres généraux

Les registres généraux peuvent être utilisés par les programmeurs en assembleur par exemple ou par des compilateurs dans le cas des programmes développés dans des langages évolués.

On dispose généralement d'une dizaine à une centaines de registres généraux. Plus le nombre de registres augmente, moins les programmes font appel à la mémoire centrale. Cependant, avoir plus de registres dans le processeur implique plusieurs circuits qui doivent prendre de la place. Les programmeurs essayent toujours d'utiliser le maximum nombre de registres possible pour éviter l'accès à la mémoire afin de maximiser les performances.

Nous avons vu dans le module 2, la différence dans la représentation des nombres entiers et flottants. Pour prendre en compte ces deux représentations, il existe des registres dédiés à chaque représentation. De la même façon, il existe une UAL dédiée pour chaque représentation, i.e. une UAL pour traiter les nombres entiers et une autre pour traiter les nombres flottants. Chaque ensemble de registres est situé près de l'UAL correspondante.

3.2 Registre d'instruction

Ce registre spécialisé, appelé aussi RI ou IR: *Instruction Register*, a pour rôle de mémoriser l'instruction à exécuter. Cette instruction est récupérée de la mémoire centrale. Le séquenceur s'en sert de ce registre pour savoir les bits qui composent l'instruction pendant l'exécution.

[3] Définition tirée du livre d'architecture de l'ordinateur d'Emmanuel Lazard.

Le programmeur n'a pas accès au registre d'instruction. C'est le séquenceur qui gère entièrement l'utilisation de ce registre spécialisé.

3.3 Compteur ordinal

Ce registre a pour rôle de mémoriser la prochaine instruction à exécuter et son adresse. Les instructions sont exécutées les une après les autres. Ce registre est appelé aussi PC: *Program Counter*, ou IP pour *Instruction Pointer*.

Donc, pour exécuter une instruction, le processeur envoie une commande de lecture mémoire à l'adresse contenue dans le PC avant de récupérer l'instruction. Ensuite, le processeur incrémente le PC de la taille de l'instruction pour que le registre pointe sur l'instruction suivante comme le montre la figure 3[4].

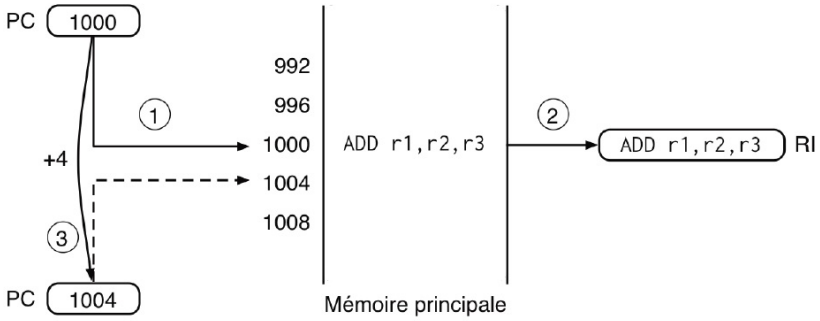

FIGURE 3 – Exemple d'incrémentation de PC.

3.4 Registre d'état

Les bits stockés dans ce registre servent d'indicateurs, appelés *drapeux* ou *flags* sur l'état du processeur. Ces indicateurs, qu'on trouve dans la plus part des processeurs, sont décrits comme suit:

- **Le bit Z (Zero)**: est mis à 1 si le résultat de l'instruction est nul, et mis à 0 sinon.
- **Le bit C (Carry)**: est mis à 1 si une retenue est générée par l'instruction, et mis à 0 sinon.
- **Le bit N (Negative)**: est mis à 1 si le résultat de l'instruction est négatif, est mis à 0 sinon.
- **Le bit V (oVerflow)**: est mis à 1 si un débordement est généré par l'instruction, est mis à 0 sinon.

[4]Figure tirée du livre d'architecture de l'ordinateur en référence.

3.5 Registre pointeur de pile

Ce registre, appelé SP: *Stack Pointer*, permet de pointer la pile pour stocker des données ou des adresses selon le principe du "Dernier Entré Premier Sorti" ou "LIFO" (Last In First Out).

4 Architecture RISC vs CISC

Les processeurs ont progressé en complexité pat l'intégration assez poussée des transistors afin d'offrir des jeux d'insertions plus complets répondant aux besoins de calculs performants.

Il existe deux architectures selon le jeu d'instruction. L'architecture classique CISC (Complex Instruction Set Computer), et la nouvelle architecture RISC (Reduced Instruction Set Computer). La différence entre ces deux architectures peut être résumée dans le tableau suivant:

Table 1: Différence entre l'architecture RISC et CISC.

RISC (Exemple de SPARC de Sun)	**CISC (Exemple de Processeur Intel)**
Grand nombre de registres	Petit nombre de registres
Petit nombre d'instructions	Grand nombre d'instructions
Peu de mode d'adressage	Beaucoup de modes d'adressage
Instruction de longueur unique	Instruction de longueur variable
Accès à la mémoire limité à certaines instructions	Accès à la mémoire pour toutes les instructions